Converting MS Basic Random Files

by *Thomas E. Kurtz*

Both MS Basic and True BASIC have random access files. By random access we mean the ability for the programmer to specify a particular record in the file, which need not be "close to" the last record used, and the ability to change the contents of a particular record without affecting other records in the file.

The simplest file type is a TEXT file, the use of which is acceptable if (a) you need only to access the information sequentially, and (b) you don't need to change stuff in the middle of the file. (You can, of course, always add stuff to the end of a TEXT file.)

While the concept of random access files is similar in MS Basic and True BASIC, the details are quite different. For this reason we made to attempt to include converting the random-file code in MS Basic into usable True BASIC code.

For simplicity reasons, this note treats MS Basic files containing only fixed-length strings.

WARNING: This writer used MSBasic documentation as a reference for information in this article, and did not personally verify the correctness of the MS Basic code. Comments are welcome.

The Differences

There are three major areas of difference between MS Basic random files and True BASIC random access file, besides, of course, minor synticatical differences in how you open and close such files.

1. In MS Basic one "moves" a file record into a record buffer in the program with a GET statement, and does the reverse with a PUT statement. In True BASIC, one "reads" a record with a READ statement, writes a record with a WRITE statement; there is no separate concept of a record buffer.

2. In MS Basic one defines "fields", which are sections in a record. These fields have fixed lengths. Obviously, the total length must not exceed the usable length of the record. In True BASIC there is no concept of fields. The content of a record is an arbitrary sequence of strings and numbers. The individual strings may be of any length so long as the total length of the contents of the record does not exceed the specified record length.

3. In MS Basic the fields of the record currently in the record buffer are given variable names. Accessing the values in the fields is done simply by using these variable names in LET statements. (They also include the LSET and RSET statements to assign string values to fields with left justification or right justification, padding with spaces. In True BASIC one simply includes the values to be read as variables in the READ statement, or as values in the WRITE statement. (In MS Basic, all records must have the same fields; while this is not necessary in True BASIC, It is the programmer's responsibility to remember which items are in which records.)

A further complication is that the rules for files in MS Basic on the Macintosh may differ from the rules for the same on DOS, and also for Visual Basic on Windows.

An Example

Suppose your random file named "members.dat" contains information on members of a club. Suppose this information includes name, address, town, state, and zip. Here is how you might do it in Macintosh QuickBasic 1.0:

```
OPEN "members.dat" AS #2, LEN = 57
FIELD #2, 20 AS name$, 20 AS address$, 10 AS town$, 2 AS
  state$, 5 AS zip$
```

to get at the information in a record, you must first GET the record, and then you can access the fields in the record by using the variable names in the FIELD statement.

```
GET #2, recordnumber
! The fields are now in the variables name$, address$,
  town$, state$, zip$
```

Of course, you don't have to "copy" this information into new variables; you can just use the information directly, as in

```
IF state$ = "NH" then      ! Must be in New Hampshire
```

To change a record, you first make sure that the correct information has been assigned to the five variables named in the FIELD statement, such as by using either LET, LSET, or RSET, followed by a PUT

```
LSET address$ = <new address>
PUT #2, recordnumber
```

If the record number happens to be after the end of the file, a new record will be added to the file.

Here is how you might do it in True BASIC

```
OPEN #2: name "members.dat", ORG random, recsize 82
```

followed by

```
SET #2: record recordnumber
READ #2: name$, address$, town$, state$, zip$
```

Later, after updating the values (for instance, a particular member may have changed her address,)

```
LET address$ = <new address>
SET #2: record recordnumber
WRITE #2: name$, address$, town$, state$, zip$
```

What to Watch For

Remember that in MS Basic the fields are fixed length. If the contents are short, the unused space will be "padded" with blank space. If the contents are longer, they will be truncated. In True BASIC the fields are all variable length, so no blank padding and no truncation will be done. However, the total length of all strings and numbers in a record must not exceed the specified record length. (The record length is specified when creating a new random file. If the random file already exists, True BASIC will pick up the record length from the file header.)

Remember that in True BASIC you must provide variables in a READ statement that correspond to the contents of the record. You can specify fewer variable names than there are contents in the record by including a IGNORE REST clause.

```
READ #2: name$ IGNORE REST
```

Otherwise, the number of variable names in the READ statement must be the same in number and type as the contents of the record. (A field can be numeric, in which case the variable that field is read into must be a numeric variable. Numeric fields occupy eight bytes.)

The Actual Length

In MS Basic the actual length of the record is what you specify in the OPEN statement, which length must be consistent with the subsequent FIELD statement for that file. In the case above, each record contains 57 bytes. If there are 100 records in the file, it will occupy 5700 bytes of disk space.

In True BASIC the actual length of the record is somewhat longer than what you specify in the OPEN statement. Furthermore, the contents of the record contain more than the actual length of the numbers (8 bytes) and strings. Each "field" in the record is preceded by one byte giving its type "N" for number and "S" for string. In addition, for strings, the next three bytes give the actual length of the string, followed by an "E" to indicate the end of the record. This comes to 78 if exactly the same information is contained in the file. If there are 100 records in the file, it will occupy a bit more than 7805 bytes of disk space inasmuch as there is a file header of five bytes containing such information as its record length.

Since names can be quite long, and the same for addresses, the wise programmer would check to see that they are limited to, say, 20 characters by using the substring expression

```
LET name$ = name$[1:20]
```

This would happen automatically in MS Basic, which truncates entries that are too long.

Processing MS Basic Random Files from True BASIC

It may sometimes be necessary to take a file created by MS Basic and process it in True BASIC without reformatting. One approach is to create two subroutines, Get and Put, that mimic the GET and PUT statements of MS Basic.

Suppose we have a file whose contents are as given earlier. The first step is to open the file with organization BYTE; this allow True BASIC to examine each byte without regard to any hidden information that may be in the file.

```
OPEN #2: NAME "MyData.dat", ORG byte, RECSIZE recsize
```

where `recsize` is the record size of the MS Basic file.

Now, the **Get** subroutine:

```
SUB Get (rn, name$, address$, town$, state$, zip$)

     ! Here is the Field statement:  FIELD #2,
     ! 20 AS name$,
     ! 20 AS address$,
     ! 10 AS town$,
     !  2 AS state$,
     !  5 AS zip$

     ! First, we set to the desired 'record'

     SET #2: record 1 + (rn-1)*recsize

     ! Next, we read in the entire record as a string.

         READ #2, bytes recsize: record$

     ! Finally, we unravel the string.

     LET name$    = record$[ 1:20]
     LET address$ = record$[21:40]
     LET town$    = record$[41:50]
     LET state$   = record$[51:52]
     LET zip$     = record$[53:57]

      END SUB
```

Finally, the **Put** subroutine.

```
SUB Put (rn, name$, address$, town$, state$, zip$)

  ! Here is the Field statement:  FIELD #2,
  ! 20 AS name$,
  ! 20 AS address$,
  ! 10 AS town$,
  !  2 AS state$,
  !  5 AS zip$
```

```
    ! First, we pad and/or truncate

    LET spaces$ = repeat$(" ",30)  ! A string of blanks
    LET n$ = (name$     & spaces$)[1:20]
    LET a$ = (address$  & spaces$)[1:20]
    LET t$ = (town$     & spaces$)[1:10]
    LET s$ = (state$    & spaces$)[1:2]
    LET z$ = (zip$      & spaces$)[1:5]

    ! Next, we set to the desired record

    SET #2: record 1 + (rn-1)*recsize

    ! Finally, we write out as the contents

    WRITE #2: n$, a$, t$, s$, z$

END SUB
```

A Word of Warning

MS Basic, particularly the more modern versions, allow other than fixed length strings as the contents of a RANDOM file. For example, one can GET or PUT an entire data structure (which is often also called a 'record'). If the data structure consists of fixed-length strings and ordinary numbers, the procedures outlined above apply. But one can also use variable-length strings, and any one of several types of numeric values. In these cases, the actual contents in the file are, as in True BASIC, preceded by a header that defines the value type, and, in the case of variable-length strings, a two-byte sequence that gives the length of the string. You should consult your reference manual for MS Basic, if you still have it!

Superficial experimentation with Visual Basic showed that writing a variable length string to a random file produced the following:

byte 1:	the length of the string in bytes (binary)
byte 2:	0, which (perhaps) is the VarType for a variable length string
byte 3 et seq:	the string itself

I can't guarantee this information as I have little experience with MS Basic or Visual Basic, but you can experiment with the program in the following section to learn the format of a file of interest.

Examining an Unfamiliar File

One can often determine the actual format of an unfamiliar file by reading it in True BASIC as a BYTE file, and examining its contents byte by byte. Perhaps this small utility will help:

```
OPEN #1: name "the file", ORG byte
DO WHILE MORE #1
   READ #1, bytes 50: s$
   FOR i = 1 to len(s$)
      LET c$ = s$[i:i]   ! The i-th byte of the string
      IF " " <= c$ and c$ <= "~" then    ! If printable, print
         PRINT c$;
      ELSE
         LET c = ord(c$)         ! Otherwise,
         PRINT "[" & str$(c) & "]"; ! print its value
      END IF
   NEXT i
   PRINT

LOOP

END
```

If a particular byte happens to be an ASCII printable character, it is printed as such; otherwise, its numeric value is enclosed in brackets and printed. With this tool you should be able to examine most types of foreign files to learn their structure.

Thomas E. Kurtz, June 1, 1999

Comments or questions to: tom@truebasic.com